

Code Formatting


Starting with the post-release 3.0 codebase, we are trying to unify the coding style of our source code. All new patches and contributions should strive to follow the same style ([See here](#) for good reasons why this is important)

Indentation: Required to be using 4 characters per level, no tabs allowed. Set your editor to insert spaces for tabs.

Comments: Self explanatory code with descriptive naming of methods and variables is preferred over comments. Only use comments if you feel the intent of your code or the algorithm used is not obvious from a casual reading of the code.

A general sample of the preferred style follows below. Follow this template for whitespace around braces, parentheses, operators and control statements.

For **Eclipse** users there is a Code Formatter template available at [opencpn:files:codestyle.xml](#) (Remove the .doc extension)

For **Uncrustify****users there is an uncrustify.cfg available at [opencpn:files:beautify.cfg.doc](#) (Remove the .doc extension)  file needs to be found. This is just a placeholder file.

If possible change editor defaults to remove blank at the end of the line. Avoid encoding, whitespace and formatting issues.

CMake Formatting

Having a uniform formatting enhances readability. The main CMakeLists.txt + some (not all) support files has been successfully reformatted using https://github.com/cheshirekow/cmake_format. The configuration file (almost pristine) is available as <https://raw.githubusercontent.com/OpenCPN/OpenCPN/master/.cmake-format.yaml>.

All in all this has created a much more consistent formatting of the CMake setup files. It might make sense to consider also for a template project

```
#include <math.h>

class Point {
public:
    Point( double x, double y ) :
        x( x ), y( y )
    {
    }
}
```

```
double distance( const Point& other ) const;
int compareX( const Point& other ) const;
double x;
double y;
};

double Point::distance( const Point& other ) const
{
    double dx = x - other.x;
    double dy = y - other.y;

    return sqrt( dx * dx + dy * dy );
}

int Point::compareX( const Point& other ) const
{
    if( x < other.x ) {
        return -1;
    } else
        if( x > other.x ) {
            return 1;
        } else {
            return 0;
        }
}

namespace F00 {
int foo( int bar ) const
{
    switch( bar ){
        case 0:
            ++bar;
            break;
        case 1:
            --bar;
        default: {
            bar += bar;
            break;
        }
    }
}
} // end namespace F00
```

From:

<https://opencpn.org/wiki/dokuwiki/> - **OpenCPN Manuals**

Permanent link:

https://opencpn.org/wiki/dokuwiki/doku.php?id=opencpn:developer_manual:developer_guide:code_formatting

Last update: **2019/12/10 18:22**

